23장 실행 컨텍스트

소스코드의 타입

ECMAScript 사양은 소소크드를 4가지 타입으로 구분

각각의 코드는 실행 컨텍스트를 생성한다.

그림 23-1 소스코드를 평가하여 실행 컨텍스트를 생성한다.

https://velog.io/@jjinichoi/모던-자바스크립트-Deep-Dive-23장-실행-컨텍스트

1. 전역코드
    - 전역 스코프 생성
    - var로 선언된 전역변수와 함수선언문으로 선언된 전역함수를 전역객체와 연결
    - 전역 실행 컨텍스트 생성
2. 함수코드

- 지역 스코프 생성
- 지역스코프를 스코프체인의 일원으로 연결
- 함수 실행 컨텍스트 생성

3. eval 코드
    - strict mode 에서 독자적인 스코프 생성
    - eval 실행 컨텍스트 생성
4. 모듈코드
    - 모듈 스코프 생성
    - 모듈 실행 컨텍스트 생성

소스코드의 평가와 실행

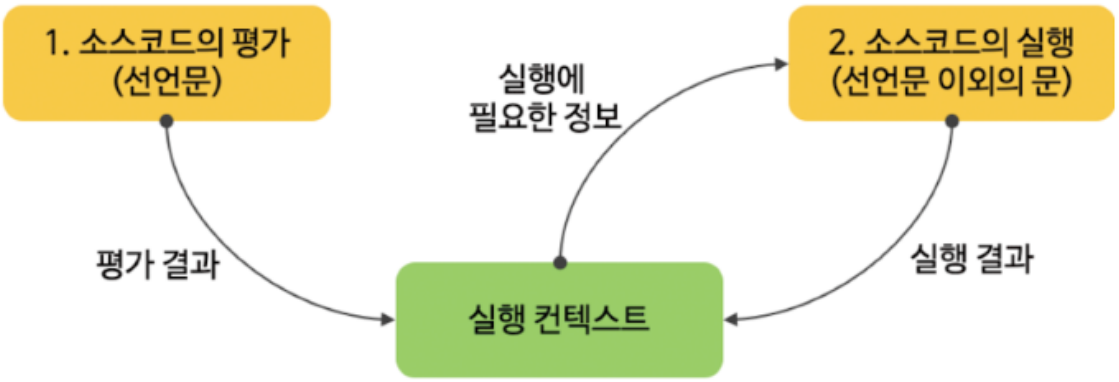

그림 23-2 소스코드의 평가와 실행

https://velog.io/@jjinichoi/모던-자바스크립트-Deep-Dive-23장-실행-컨텍스트

1. 소스코드의 평가
    a. 실행 컨텍스트 생성
    b. 선언문 실행
    c. 실행컨텍스트의 스코프에 등록

2. 소스코드의 실행
    a. 런타임 실행
    b. 실행 결과 실행컨텍스트의 스코프에 등록

실행 컨텍스트의 역할

- 전역코드와 함수코드의 실행순서

1. 전역코드 평가
    a. 선언문 실행
    b. 전역 스코프에 등록
2. 전역코드 실행
    a. 런타임 시작
    b. 전역변수에 값이 할당, 함수 호출
    c. 함수 호출시 전역코드 실행 중단 일시 중단 후 해당 함수 내부로 이동
3. 함수코드 평가
    a. 매개변수와 지역변수 선언문 실행
    b. 지역 스코프에 등록
4. 함수코드 실행
    a. 런타임 시작
    b. 함수코드 종료 후 전역 코드 실행 재개

이러한 코드 실행 순서를 위해선 3가지 역할이 필요하다

1. 모든 식별자를 스코프를 구분하여 등록하고 상태변화를 지속적으로 관리
2. 스코프 체인 형성 및 스코프체인 내의 식별자 검색
3. 코드의 실행 순서 변경 및 원복가능

실행컨텍스트 는 필요한 환경을 제공하고 코드의 실행결과를 실제로 관리하는 영역

모든 코드는 실행컨텍스트를 통해 실행되고 관리된다

실행 컨텍스트 스택

실행컨텍스트는 생성 후 스택 자료구조로 관리되는데 이를 실행컨텍스트 스택이라 한다

논리적 스택 구조를 가지는 실행 컨텍스트 스택

https://hong-p.github.io/javascript/javascript-deepdive-ch23/

위의 예시에 따라 코드가 실행되는 시간의 흐름에 따라 추가(push)되고 제거(pop)된다.

실행 컨텍스트 스택은 코드의 실행순서를 관리한다

실행컨택스트 실행 순서에 따르면 실행컨텍스트 스택의 최상위에 존재하는 실행컨텍스트는 언제나 현재 실행 중인 코드의 실행 컨텍스트이다

이를 실행중인 실행 컨텍스트라 부른다

렉시컬 환경

자료구조로 실행 컨텍스트를 구성하는 컴포넌트이다.

실행 컨텍스트 스택은 코드의 실행 순서를 관리하고

렉시컬 환경은 스코프와 식별자를 관리한다

렉시컬환경의 역할

1. 키와 값을 갖는 객체형태의 스코프를 생성하여
2. 식별자를 키로 등록하고
3. 식별자에 바인딩된 값을 관리한다.

렉시컬환경 의 구성

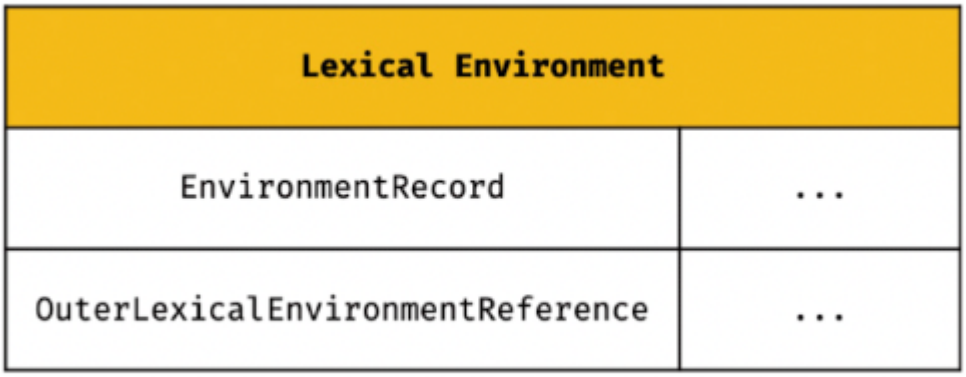

그림 23-8 렉시컬 환경의 구성 컴포넌트

https://velog.io/@jjinichoi/모던-자바스크립트-Deep-Dive-23장-실행-컨텍스트

1. **환경 레코드 (Environment Record)**
   a. 식별자를 등록하고 등록된 식별자에 바인딩된 값을 관리
2. **외부 렉시컬 환경에 대한 참조 (Outer Lexical Environment Refernece)**
   a. 상위 스코프
   b. 상위 코드의 렉시컬 환경

실행 컨텍스트의 생성과 식별자 검색과정

```
car x = 1;
const y = 2;

function foo (a) {
  var x =3;
```

```
  const y = 4;

  function bar (b) {
    const z = 5;
    console.log(a + b + x + y + z);
  }
  bar(10);
}

foo(20);  // 42
```

1. 전역객체 생성
    - 전역 코드 평가 이전 생성
2. 전역코드 평가
    a. 전역 실행 컨텍스트 생성
    b. 전역 렉시컬 환경 생성
        i. 전역 환경레코드 생성
            1. 객체 환경 레코드 생성
            2. 선언적 환경 레코드 생성
        ii. this 바인딩
        iii. 외부 렉시컬 환경에 대한 참조 결정

전역코드의 평가

1. 전역 실행 컨텍스트 생성

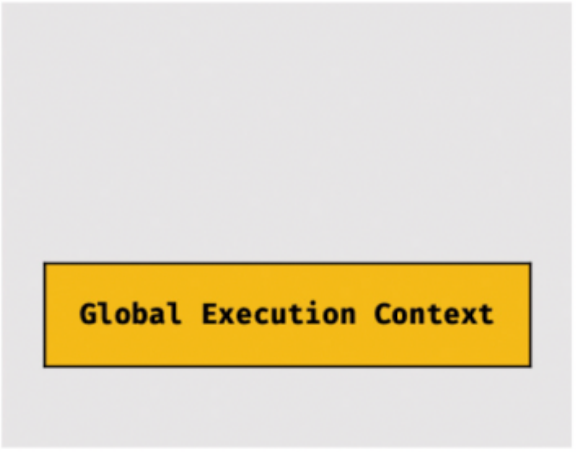

그림 23-10 전역 실행 컨텍스트 생성

https://velog.io/@jjinichoi/모던-자바스크립트-Deep-Dive-23장-실행-컨텍스트

- 전역 실행 컨텍스트를 생성하여 실행컨텍스트 스택에 푸시
- 전역 실행 컨텍스트가 실행중인 실행 컨텍스트가 된다

2. 전역 렉시컬 환경 생성

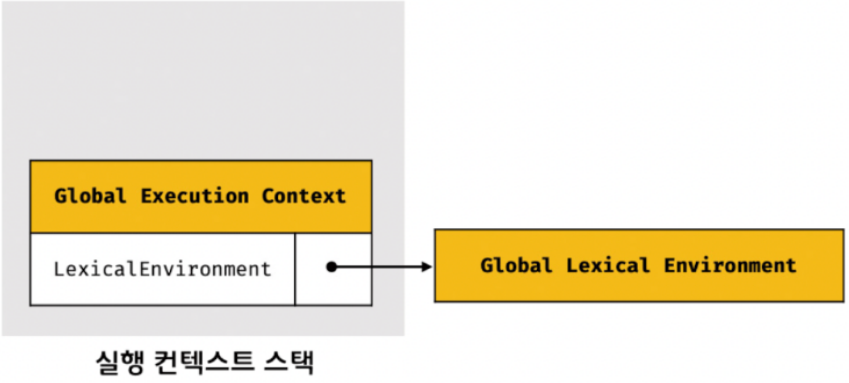

그림 23-11 전역 렉시컬 환경 생성

https://velog.io/@jjinichoi/모던-자바스크립트-Deep-Dive-23장-실행-컨텍스트

- 전역 렉시컬 환경을 생성하고 전역 실행 컨텍스트에 바인딩한다.
- 렉시컬 환경은 2개의 컴포넌트로 구성된다

  2.1 전역 환경 레코드 생성

- 전역 변수를 관리하는 전역스코프 이다.
- 전역 환경 레코드는 객체환경레코드와 선언적 환경레코드로 구성되어있다

2.1.1 객체 환경 레코드

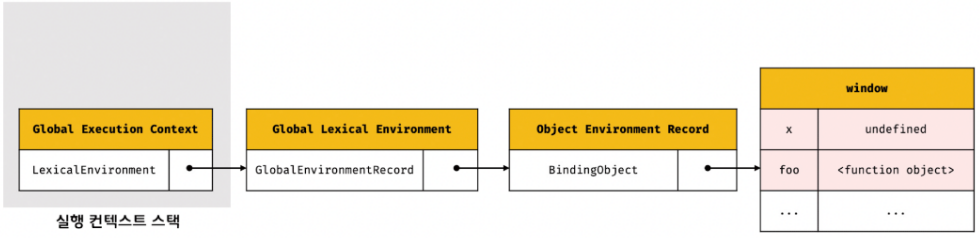

그림 23-12 전역 환경 레코드의 객체 환경 레코드

https://velog.io/@jjinichoi/모던-자바스크립트-Deep-Dive-23장-실행-컨텍스트

- 전역 변수와 전역함수 및 표준 빌트인 객체를 관리
- 객체 환경 레코드는 BindingObject객체와 연결되어 있다
- BindingObject 를 통해 전역객체의 프로퍼티와 메서드가 된다
  (전역객체 식별자 없이 전역변수를 호출할 수 있는 이유)

2.1.1 선언적 환경 레코드

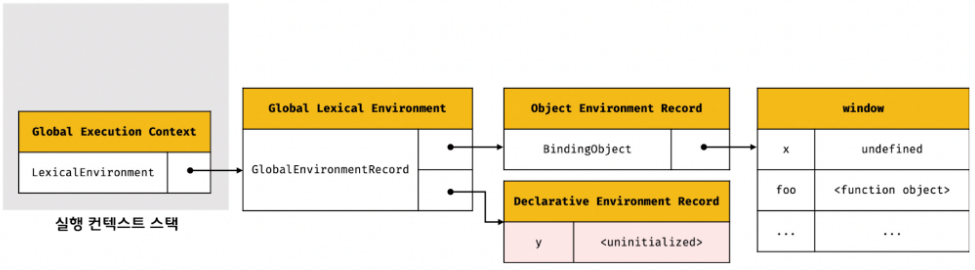

그림 23-13 전역 환경 레코드의 선언적 환경 레코드

https://velog.io/@jjinichoi/모던-자바스크립트-Deep-Dive-23장-실행-컨텍스트

- let,const 키워드로 선언한 전역 변수를 관리

2.2 this 바인딩

전역 환경 레코드의 [[GlobalThisValue]] 내부 슬롯에 this 바인딩

전역코드에서는 전역객체를 가르킨다

this 바인딩은 전역 환경 레코드와 함수 환경 레코드에만 존재한다.

2.3 외부 렉시컬 환경에 대한 참조 결정

전역 레시컬 환경의 외부 렉시컬 환경에 대한 참조는 null 이 할당된다

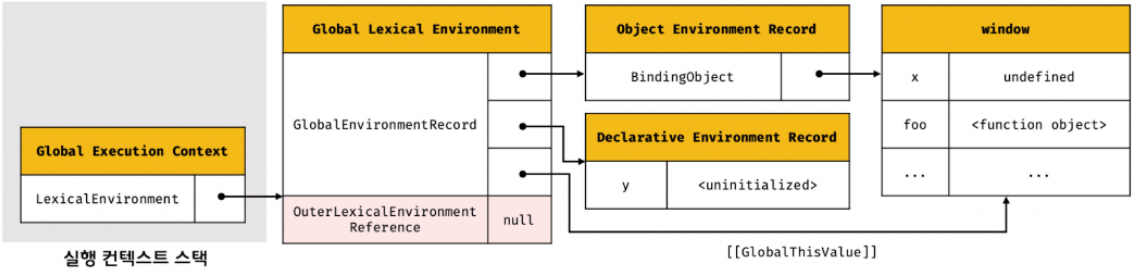

그림 23–15 외부 렉시컬 환경에 대한 참조 결정

https://velog.io/@jjinichoi/모던-자바스크립트-Deep-Dive-23장-실행-컨텍스트

이는 스코프체인의 종점에 존재함을 의미한다.

전역코드의 실행

전역 코드가 순차적으로 실행

식별자 결정

- 변수 할당문 또는 함수 호출문을 실행하려면 먼저 변수 또는 함수 이름이 선언된 식별자인지 확인이 필요하다.
- 또한 식별자는 스코프가 다르면 같은 이름을 가질수 있다. 따라서 어느 스코프의 식별자를 참조하면 되는지 결정할 필요가 있다. 이를 **식별자 결정**이라한다.
- 식별자 결정을 위해 식별자를 검색할 때는 실행 중인 실행 컨텍스트에서 식별자를 검색하기 시작한다. 만약 **실행 중인 실행 컨텍스트의 렉시컬 환경에서 식별자를 검색할 수 없으면 외부 렉시컬 환경에 대한 참조가 가리키는 렉시컬 환경(상위스코프)로 이동하여 식별자를 검색**한다.
- 이것이 바로 **스코프 체인의 동작 원리**다.

```
// go!
function foo (a) {
  var x =3;
  const y = 4;

  function bar (b) {
    const z = 5;
    console.log(a + b + x + y + z);
  }
  bar(10);
}
foo(20);  // 42
```

foo 함수 평가

1. foo 함수 실행 컨텍스트 생성
2. 함수 렉시컬 환경 생성
    - foo 함수 렉시컬 환경을 생성하고 foo 함수 실행 컨텍스트에 바인딩

    2.1 함수 환경 레코드 생성

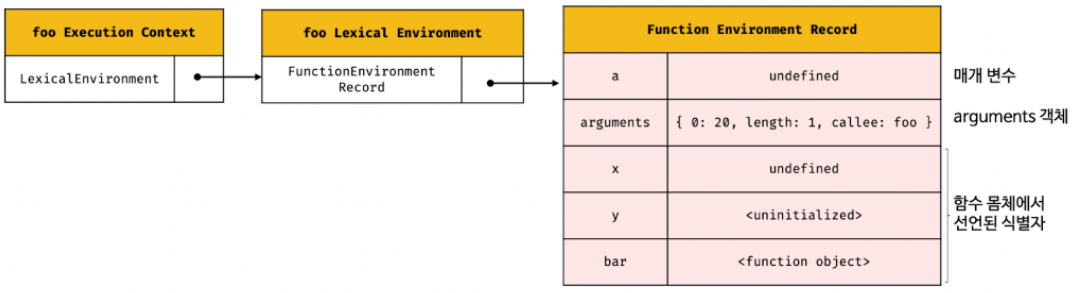

그림 23-19 함수 환경 레코드의 환경 레코드

https://velog.io/@jjinichoi/모던-자바스크립트-Deep-Dive-23장-실행-컨텍스트

- 함수 내부에서 선언한 지역변수와 중첩함수를 등록하고 관리

2.2 this 바인딩

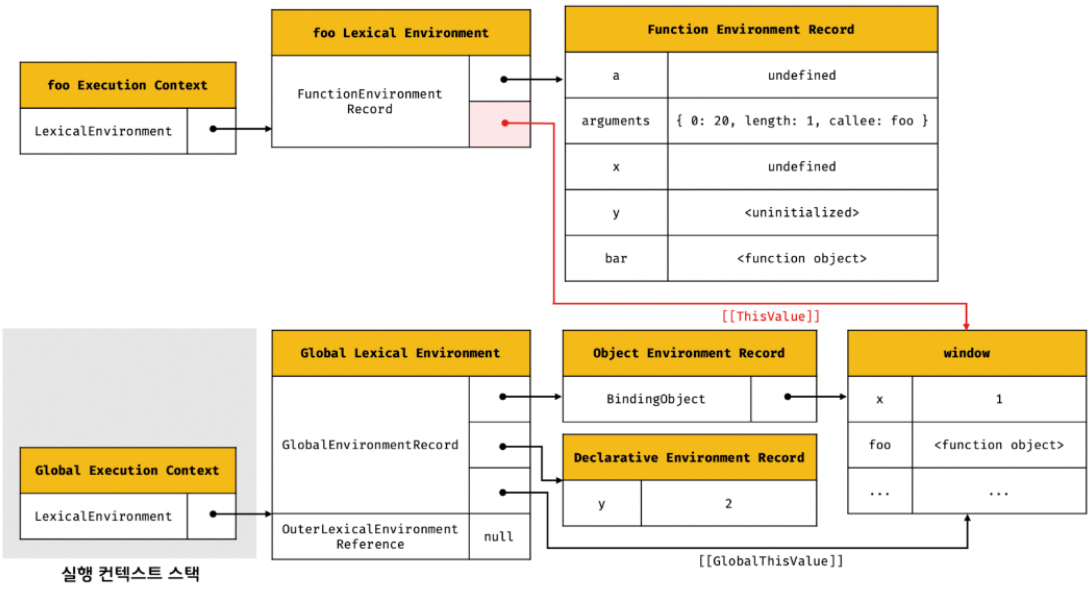

그림 23-20 this 바인딩

- foo 함수는 일반함수로 호출되었으므로 전역객체를 가리킨다.

2.3 외부 렉시컬 환경에 대한 참조 결정

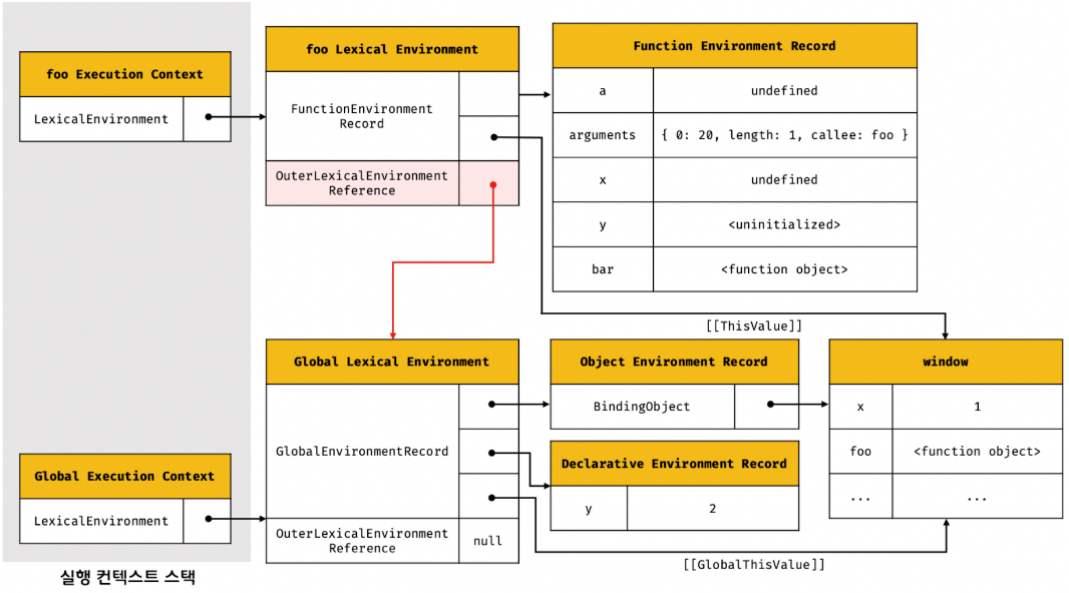

그림 23-21 외부 렉시컬 환경에 대한 참조 결정

- 전역 렉시컬 환경에 대한 참조가 할당
- 함수 렉시컬 환경의 외부 렉시컬 환경에 대한 참조에 할당되는 것은 바로 함수의 상위 스코프를 가리키는 함수 객체의 내부 슬롯 에 저장된 렉시컬 환경의 참조다. 즉,

- **함수 객체의 내부 슬롯 [[Environment]] 가 바로 렉시컬 스코프를 구현하는 메커니즘이다.**

foo 함수 실행

```
function foo (a) {
    // go!
  var x =3;
  const y = 4;

  function bar (b) {
    const z = 5;
    console.log(a + b + x + y + z);
  }
  bar(10);
}
// go!
foo(20);  // 42
```

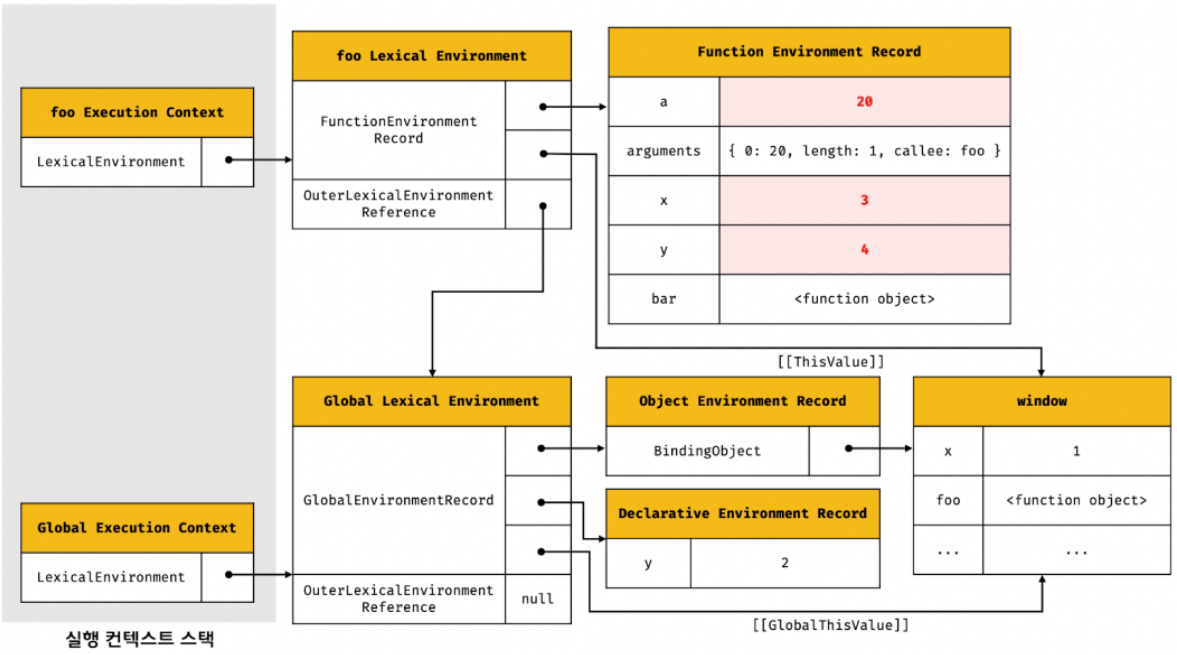

그림 23-22 foo 함수 코드의 실행

- 소스코드가 순차적으로 실행

- 식별자 결정을 위해 렉시컬 환경에서 식별자를 검색한다.

bar 함수 코드 평가

```
function foo (a) {
  var x =3;
  const y = 4;
    // go!
  function bar (b) {
    const z = 5;
    console.log(a + b + x + y + z);
  }
  bar(10);
}
foo(20);  // 42
```

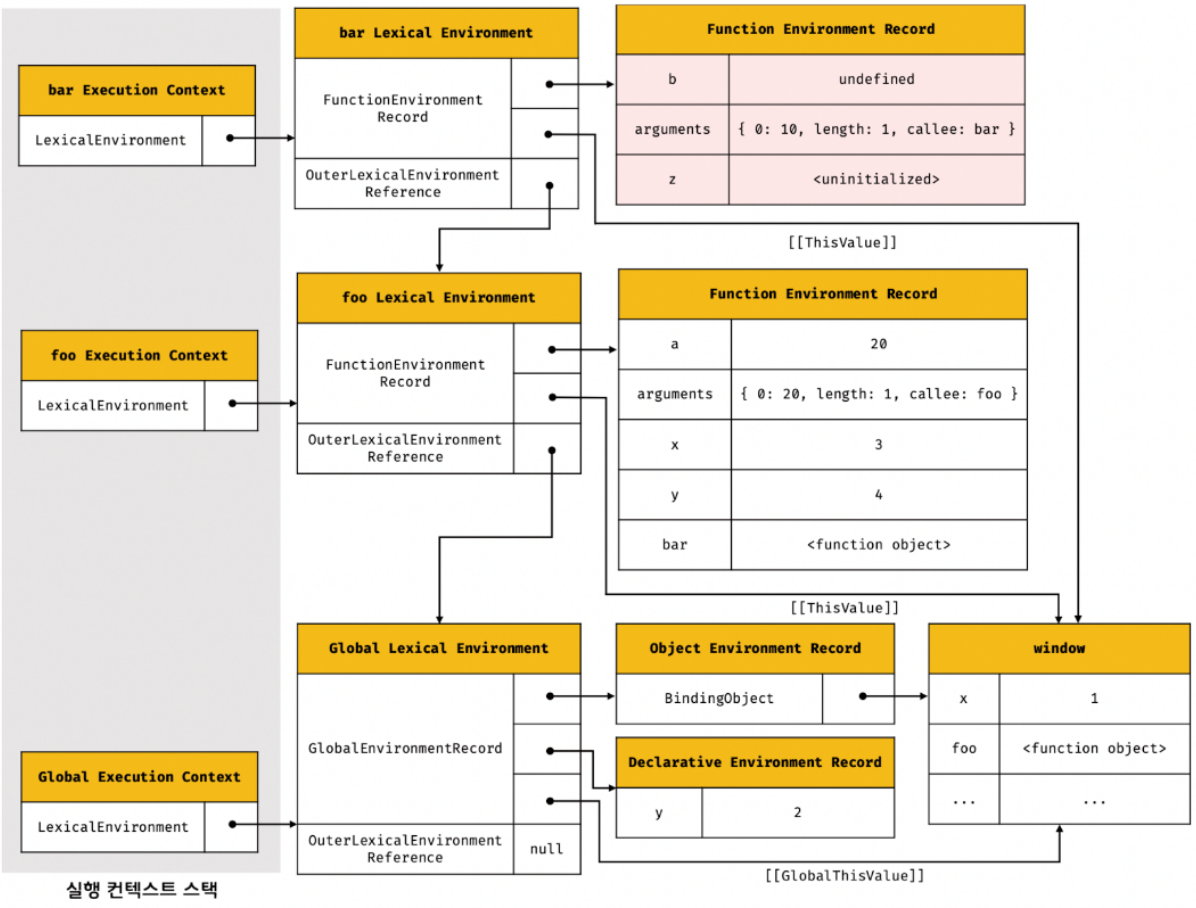

그림 23-23 bar 함수 실행 컨텍스트와 렉시컬 환경

- 동일한 과정으로 bar 함수를 평가한다

bar 함수 코드 실행

```
function foo (a) {
  var x =3;
  const y = 4;
  function bar (b) {
      // go!
    const z = 5;
    console.log(a + b + x + y + z);
  }
  // go!
  bar(10);
}
foo(20);  // 42
```

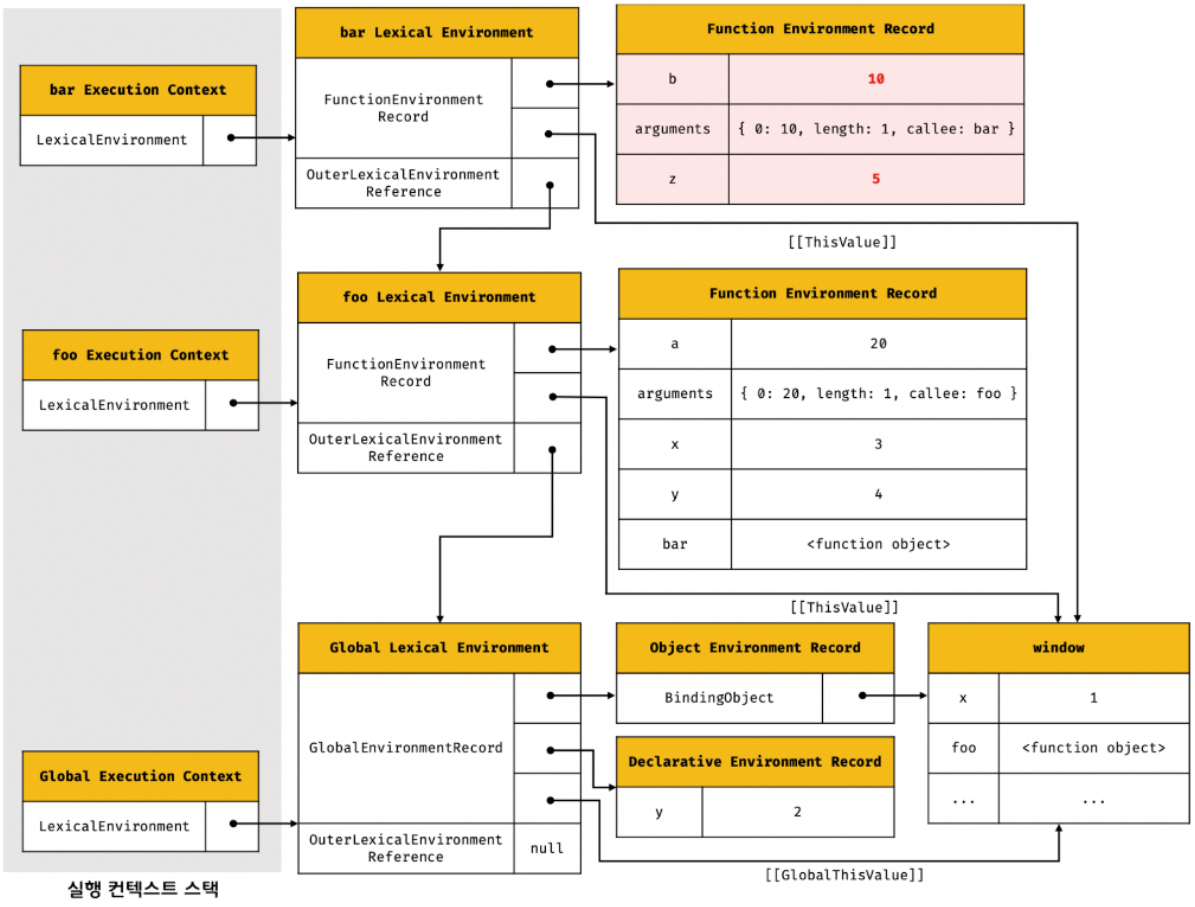

그림 23-24 bar 함수 코드의 실행

- 동일한 과정으로 bar 함수를 실행한다
- console 식별자를 스코프체인에서 검색하여 전역객체에서 찾아 사용한다.

bar 함수 코드 실행 종료

- 더이상 실행할 코드가 없으므로 실행컨텍스트가 팝되서 제거된다
  - 실행컨텍스트가 제거됨과 다르게 렉시컬 환경은 독립적인 객체로써 즉시 소멸되지 않는다
  - 누군가에의해 참조되지 않을때 가바지 컬렉터에 의해 소멸한다.
- foo 실행 컨텍스트 이동

foo 함수 코드 실행 종료

- 더이상 실행할 코드가 없으므로 실행컨텍스트가 팝되서 제거된다

전역 코드 실행 종료

- 더이상 실행할 코드가 없으므로 실행컨텍스트가 팝되서 제거된다
- 실행 컨텍스트 스택이 비워진다

실행 컨텍스트와 블록레벨 스코프

- let, const 키워드로 선언한 변수는
- 모든 코드 블록을 지역 스코프로 인정하는 블록 레벨 스코프를 따른다.

```
let x = 1;
if(true) {
  let x = 10;
  console.log(x); // 10
}
console.log(x); //1
```

- if 문의 코드 블록이 실행되면 if문의 코드 블록을 위한 블록 레벨 스코프를 생성해야 한다.
- 이를 위해 선언적 환경 레코드를 갖는 렉시컬 환경을 새롭게 생성하여 기존의 전역 렉시컬 환경을 교체한다.
- 이때 새롭게 생성된 if 문의 코드 블록을 위한 렉시컬 환경의 외부 렉시컬 환경에 대한 참조는 if문이 실행되기 이전의 전역 렉시컬 환경을 가리킨다.

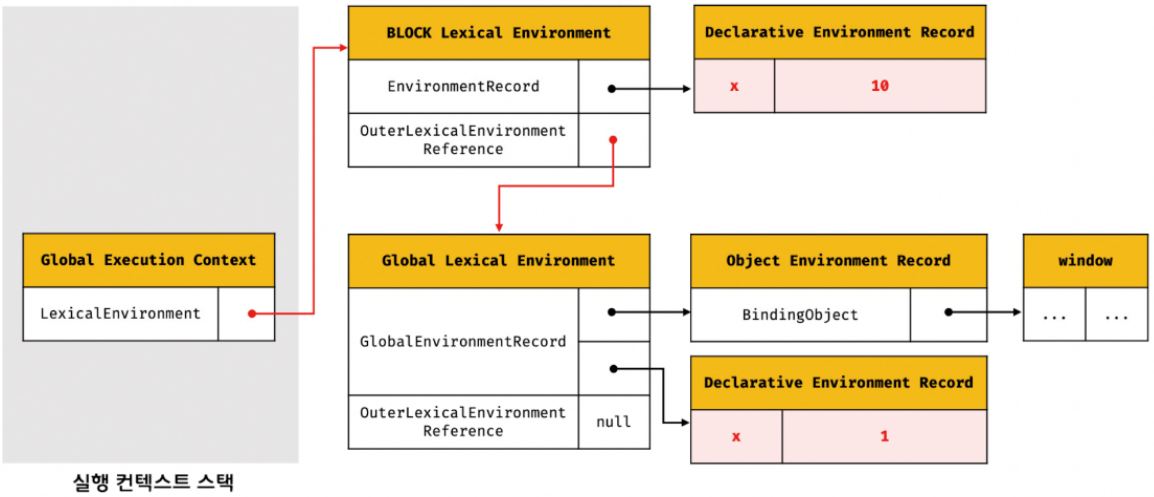

그림 23-28 if 문의 코드 블록이 실행되면 새로운 렉시컬 환경을 생성하여 기존의 렉시컬 환경을 교체